マルチメディア通信と分散処理 53－6
（1992. 1. 23）

分散コンピューティング環境(DCE)

オープンシステム時代における分散処理環境のインフラストラクチャ

永島道夫
日本ディジタル イクイップメント株式会社
オープンシステム事業部 製品企画部

性能的にも機能的にも実用に供することが可能な，しかも移植性の保証された分散処理システムならびに分散アプリケーション構築の要求が高まってきている。初めに，この要求に標準化動向が従来どの様に対応してきたかを概括する。つぎにこの発展として，オブジェクト指向分散マルチメディア環境の実装について述べる。更に，より広範な要求に応えるためのインフラストラクチャとなるべく*1991*年*10*月に出荷が開始された分散コンピューティング環境，*OSF/DCE(Distributed Computing Environment)*について，その概要を述べる。ただし，全体系は極めて複雑，且つ，多岐にわたるので，全体を満遍無く述べることよりも特定の話題に的を絞っての記述を試みた。

Distributed Computing Environment(DCE)

Infrastructure for distributed information processing in the era of Open Systems

Michio Nagasahima
Product Marketing, Open Systems Business Group
Digital Equipment Corporation Japan,

Demands for establishing portable Distributed Information Processing and Applications, with enough functionality and performance, become common among the industries. Standardization efforts have been responding them with various style of implementation.

One of the hottest theme of today is the implementation of Object-Oriented Distributed Multimedia Computing Environment.

In order to establish more generarized distributed information processing, OSF has defined its specification, productized it and begun to ship OSF/DCE(Distributed Computing Environment) in October 1991.

1. 歴史的動向

まず，アプリケーション実行環境の実装技術，環境テクノロジの歴史的動向を概括する（表１）．

表１ 環境テクノロジの発展形態

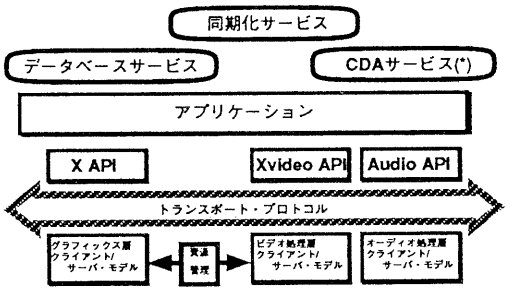
環境テクノロジ	従来テクノロジ	現在および将来	備考
ウインドウシステム	各社固有のウインドウシステム	X Window System	MIT開発
ファイルシステム	各社固有のファイルシステム	NFS AFS(OSF/DCE)	Sun Micro OSF開発
処理能力の共有	各社固有のプロセス間通信	NCS/RPC (OSF/DCE)	OSF開発
システム管理	各社固有の管理システム	OSF/DME	OSF開発
マルチメディア	各社固有のマルチメディア体系	分散統合環境化 マルチメディア	標準化 相互運用性

一般的に環境テクノロジはローカル型から分散型へと発展することが容易に分る．ウインドウ・システムにおけるX，分散ファイルシステムのＮＦＳ等がその例である。更に，現在商業的に最もホットな話題のひとつであるマルチメディアについても同様に，標準化・相互運用性の確立による，分散化の方向性において発展すると思われる。

2. 分散マルチメディアコンピューティング環境

この方向性におけるマルチメディアの実現の一例として，DECにおける分散化マルチメディア体系のインプリメンテーションを次に示す。

図１ 分散マルチメディア体系

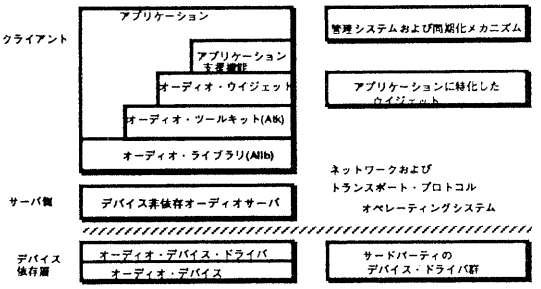


(*)CDA : Compound Document Architechture

マルチメディア・アプリケーションは，音声・画像同期化サービス，マルチメディア対応DBMSサービス，複合ドキュメント・サービス等の支援の下に３種のAPIを利用して，マルチメディア機能を実現する。分散マルチメディア環境における固有のAPIはXvideo API および Audio APIの２種であり，これらは，X Window Systemがそうであったように，物理層に依存しないトランスポート機構により，分離され，クライアント／サーバ・スタイルのプログラミング・パラダイムを提供する。

実際，例えばオーディオ処理のプログラミングモデルは次の図のようになる。

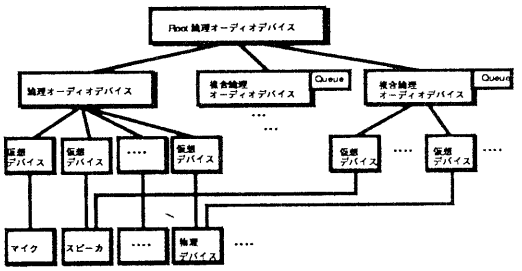
図２ 分散化オーディオ処理体系モデル



体系は，クライアント側とサーバ側に別れたクライアント・サーバ型である。クライアント側のAPIは下位レベルのライブラリAlibから上位のツールキットAudioToolKit(Atk), 更にOSF/Motif環境でのプログラミングの容易さのためにAudio Widgetを準備する。一方，サーバ側もデバイス依存層と非依存層に別れ，将来的なオーディオ要素技術の発展が行なわれた場合にも，クライアントから見えるサーバのモデルの変更は最小限に押さえられる。ユーザのアプリケーション資産の継承は容易である。このプログラミング方式は正にＸに相似形となっている。

更に，アプリケーションが扱うオブジェクトの体系は次の様に定義される。

図３ 分散化オーディオ・オブジェクト体系



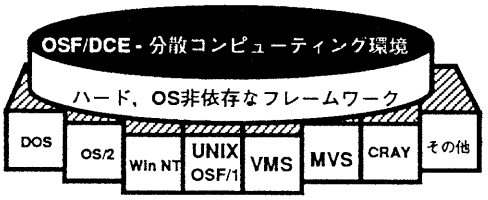
物理デバイスは仮想デバイスとしてカプセル化され，これを上位の各種論理オーディオデバイスが利用する。例えば，ビデオ会議システムの音声処理部は会議参加者数だけのインスタンスを持つ複合論理デバイスである。各参加者からの音声出力リクエストはキューから順次取り出され，通常，ワークステーションにひとつしかないスピーカにオーディオ・サーバが自動的にミキシングして出力する。分散化フレームワークはアプリケーション開発・付加価値の向上に大きな寄与する事が分かる。

3. OSF/DCE概要

OSF(Open Software Foundation)は1991年10月，公開的な技術公募によって選択されたテクノロジを基礎にOSF/DCE(Distributed Computing Environment)分散コンピューティング環境（以下 DCE）の開発キットの出荷を開始した。

この成果品はソースコードとして提供され，各システム／ソフトウエア開発者は相互運用性を損なわない範囲で自由にインプリメンテーションを行う事ができ，UNIXの枠を越えて実装が行なわれている。

図4 OSF/DCEのプラットホーム



3.1 記述の方針

OSF/DCEは極めて多くの内容を含んだ体系であり，限られた紙幅では全てを尽くすこと多は困難である。このため，全体については，大まかな方針の記述に限定し，やや詳しい解説を次のものに限定して行う。

- リモート・プロシジャ・コール(RPC)のサービス要求／提供のバインディング方式各種
- ディレクトリ・サービスにおけるローカルおよびグローバル・ネームスペースの構成
- 分散ファイルシステム(OSF/DFS)の商業分野での有効性

3.2 DCEのアプローチ

- 解決すべき課題：複数の独立したコンピューティングを一貫性を持って統合する
- ソリューション：異機種マルチベンダ環境でのアプリケーション開発，運用，保守をサポートする。
- ゴール：基本的なサービスを提供する
  - -- プロトコルとインタフェースの仕様
  - -- 参照インプリメンテーションの提供
  - -- 全体的な検証手段の提供
- ターゲット
  - -- アプリケーション・プログラマ
  - -- エンド・ユーザ
  - -- システム管理者

3.3 DCEの特長／特性

- 物理的に分散している
- 管理機能の自律性
- 異機種／異環境の共存
- 多重性
- 再構成性

3.4 DCEによって期待されること

- 資源の共有
- テクノロジーの有効利用
- 高可用性
- 性能
- 拡張可能性

3.5 DCEの技術的挑戦

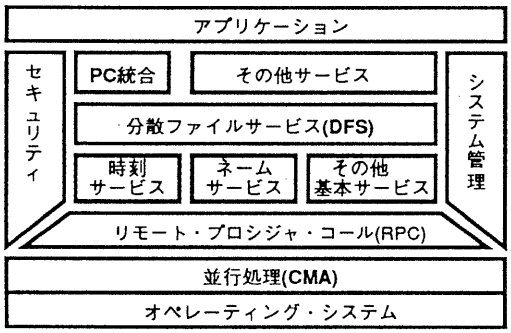
- ネットワーク透過性
- ネーミング
- セキュリティ
- スケーラビリティ
- 管理用意性
- 非決定論性
- ネットワーク・オーバヘッド

3.6 DCE設計方針

- 透過性：物理適に複雑な分散環境の上に，単純な論理層を作り上げる。
  - -- 馴染みの無いメカニズムを馴染みのあるパラダイムで隠蔽する。
  - -- 「サービスはシステムである」
- 全体的なフレームワークの提供：
  - -- 各要素の共存に留まらない体系
  - -- 通信，ネーミング，セキュリティ，時刻等の統一性
  - -- オープン・プロセスによる技術公募
  - -- メンバーとなっている組織や個人，技術提供者，企画団体，コンサルテーション等の対話的な参加
- 成熟したテクノロジーと革新的なテクノロジーの間のバランス

4. OSF/DCEのアーキテクチャ

4.1 全体アーキテクチャ



4.2 DCE - 基本構成要素

全体は次の構成要素からなる：

- スレッド
- ディレクトリ・サービス
- セキュリティ
- 時刻サービス
- 分散ファイルシステム
- PC統合サービス
- マネジメント
  - -- これらの要素はそれぞれが自律的な管理機能を持つ様にデザインされる。
  - -- 今後出荷が予定されている完全な管理アーキテクチャ（ＯＳＦ／ＤＭＥ）へ統合する様に計画されている。

また，ソフトウエアの成果品としての構成要素ではないが，「セル」の概念はDCEにおいては基本的なコンセプトの枠組みを構成している。

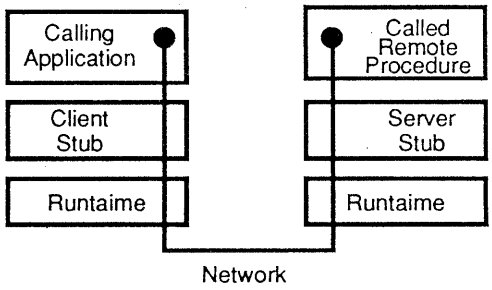
- 分散環境の組織構成原理
- 社会的，政治的，企業上の境界に従う
- 複数のセルはグローバルな空間に統合される
- セルは各種の管理のスコープ
- 次の構成要素にとって自然なドメインとなる
  - -- ネーミング
  - -- ファイル・システム
  - -- セキュリティ
  - -- 管理
- 最適化は，セル内(intra Cell)，セル間(Inter Cell)それぞれに対して行なわれる。
  - -- セル間通信
    - = 一般的に距離と遅延が大きい
    - = 多様な信頼性レベル
    - = 予測不可能なコンフィギュレーションの変化

5. RPCの概要

5.1 リモート・プロシジャ・コール

- 一般的なローカル・プロシジャ・コールのパラダイムをなるべく利用し
  - -- 自然さ
  - -- 分散動作のための特有の拡長を行う
- 通信プロトコルからの独立性により移植性を保証
- 異機種間におけるデータ表現形式の差異を吸収する
- プロシジャ間インタフェースは

5.2 RPC プログラムモデル



- 呼出し側アプリケーションはスタブにリンクされる。
- 呼ばれるアプリケーション側はスタブにリンクされる。
- 呼び出す側を「クライアント」と呼ぶ
- 呼ばれる側を「サーバ」と呼ぶ
- クライアントはインタフェースをインポートする
- サーバはインタフェースをエクスポートする
- DCE においてはインタフェースには名前と，固有のＩＤがある。このＩＤをUUID(Universal Unique Identifier)と呼び，これはuuidgen コマンドにより生成される。
- これはプログラムがRPCを仕様するために，開発者が行わなければならない，作業のひとつである。

5.3 RPC の構成要素

次の構成要素からなる。

- インタフェース定義言語コンパイラ
- 生成されたスタブ
- ランタイム

5.4 OSF/DCE RPCのバインディング解決手法

一般的には，リモート・プロシジャ・コールは手続き呼び出しの単なるネットワークへの延長として考えられるが，異なるのは，クライアントの要求に応えるサーバ側との相互の接続を決定する方法，バインディングの解決が必要となることである。この解決を行うために，クライアントとサーバをつなぐものとして「バインディング・ハンドル」，略して単に「ハンドル」を用意する。このハンドルを用いる解決方法は 3 種類用意されている。

- パラメトリック・バインディング・ハンドル
- 暗黙のハンドル指定
- 自動的ハンドル

5.5 パラメトリック・バインディング・ハンドル

この場合には各コールに明示的にハンドルがわたされる。コーディング上もリモート・コールの引き数に，ハンドルを付け加えたものになる。第 1 引き数が，handle_t となる。

- 長所
  - -- 最も柔軟性がある。ハンドルの値により，複数のサーバを利用する事が可能。プログラマにとっては最も自由度が高い。
- 短所
  - -- シンタックスがローカル・コールと異なってしまうので，ソースコード上のネットワークの構造の隠蔽性が損なわれる。

5.6 暗黙のハンドル指定

- 単一の外部ハンドルを用意する。
  [implicit] handle_t handle;
  depo (acct, amount, status);
- 長所：透過性
  - -- RPCを用いても当該コーディング部分はローカル・コールの様に見える
  - -- バインディング・ハンドルは隠蔽されている
  - -- 全てのコールは同じサーばに行く
- 短所：透過性
  - -- 1 つのインタフェースに対する複数のRPCコールは，全て同じサーバに行く。
  - -- サーバの選択はプログラマが行う。

5.7 自動的ハンドル

最も単純なバインディングであり，暗黙のハンドルによるものよりも透過性は高い。ネットワークの状況を定義する，定義ファイルをコンパイルするIDLコンパイラが全ての参照の問題を解決する。

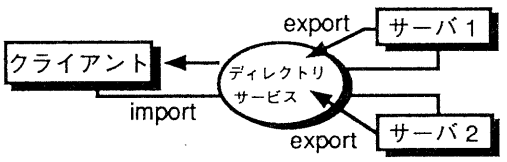
- バインディング・ルーチンを生成する
- サーバ位置とサーバの選択をスタブの中に埋め込んでしまう。
- インタフェース定義の情報を使って，サービスを提供してくれるサーバを見出す。
- アクセスしたサーバに障害が発生していてサービスが受けられない場合には自動的に再バインディングをする。
- 動作は，属性コンフィギュレーション・ファイル(ACF)で推定する。

5.8 PRCバインディングによるディレクトリ・サービスの利用

クライアントとサーバーの間の接触はこのサービスにより始まる。ディレクトリ・サービスでは，「ネーミング」，名前によってネットワーク中のオブジェクトを認識する。例えば，個々のサービスそのもの，ホストマシン，ユーザ，プリンタ・キュー，等。

- サーバーはまずディレクトリ・サービスに自分自身の情報を与える。
- クライアントは動サービスからサーバーに関する情報を得る。

図 5 サーバーの位置情報の流れ



- サーバーはインタフェースをエクスポートする。自身のバインディング情報をネーム・スペース内に置く
- クライアントは
  - -- 自分に適合するサーバーを特定する
  - -- インタフェースのインポートを行う
  - -- クライアントがコールする
  - -- 名前付きインタフェースにバインディングを取り出す。
- 適合性のあるサー羽の自動選択
  - -- binding_import

= インタフェースにハンドルを返す
= <インタフェース名>_import
= コンパイラが各インタフェースに対して生成する。

-- プログラマが選択するバインディング
-- ルックアップ - バインディングのセットを返す
-- 開発者は選択のアルゴリズムを用意する
-- 実行時の適合性のあるサーバーの選択はランダム抽出で行なわれる。

- 「プロファイル」
  -- 適合性のあるバインディングのパスを捜すのに利用するための，優先順位付きのサーバーのリストである。
- グループ
  -- 等価なサーバをグループ化する。
  -- 可用性と信頼性
- グループやプロファイルは包含関係を持つ事ができる。

5.9 RPCアプリケーションの開発

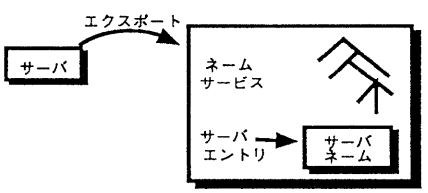
開発の手順は次の通りである：

- IDLを用いてリモート・インタフェースを定義する。
- IDLコンパイラがスタブとヘッダ・ファイルを生成する。
- アプリケーション・サーバ・ルーチンを開発する。
- スタブをコールするクライアント・コードを開発する。
- アプリケーション・コード，スタブ，およびRPCランタイムをリンクする。

5.10 RPCサーバーの動作

サーバはエクスポートするインタフェースをランタイムに通知する。

図6 サービスのエクスポート



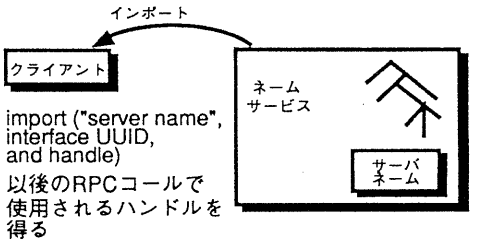
export ("server name", binding information)

- rpc_server_register_if
- リスニング・ポートをセットアップする。
- rpc_server_use_all_protseqs
- 各ポートに対してバインディング・ハンドルを取得する。
  -- rpc_server_inq_bindings
- インタフェースとバインディングをネーム・スペースにエクスポートする。
  -- rpc_ns_binding_export
- サーバー処理の依頼をリスニングする。
  -- rpc_server_listen

5.11 RPCクライアントの動作

クライアントはディレクトリ・サービス中に適合するサーバーを「捜す」。

図7 インタフェースのインポート



- インタフェースをインポートし，バインディングを取得する（この共通コード部分はクライアント開発の一部である）。
  -- rpc_ns_import_begin
  -- rpc_import_next または
- auto_handole でバインディングを選択する。
- RPCをコールする。

5.12 RPCとスレッドの併用

- サーバ
  -- 処理を別々のスレッドに割り当てる
  -- 複数の処理要求を同時に扱う
- クライアント
  -- 複数のRPCを同時に予備だセル。
  -- データ転送とデータ処理をオーバーラップして行える。
- RPCとスレッドはOSF/DCEのフレームワークの中で統合化されている。

6. ディレクトリ・サービス

6.1 ディレクトリ・サービスの目標

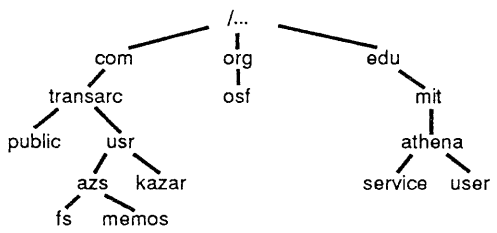
- 汎用的なネーミング
  - -- ネットワーク中の全てのオブジェクトがネーミングできる
- 名前とオブジェクト間の単一的・統一的なマッピング
  - -- 全てのオブジェクトは名前を持つ
  - -- それぞれの名前は1つのオブジェクトにマッピングされる
- 名前空間
  - -- 階層的，記憶が容易，推測が可能

6.2 統合化された命名アーキテクチャ

- グローバル・コンポーネント
  - -- 国際規格(X.500)
  - -- 大域的な相互運用性
- セル・ディレクトリ・サービス(CDS)
  - -- 高速
  - -- 可用性
  - -- 統合化容易
- 特殊化された命名システム
  - -- DFS
  - -- セキュリティ・サービス
  - -- ユーザ・アプリケーション

セル（ローカル）およびグローバルネーム空間を統合化した全体は次の様になる。

図8 統合化グローバル・ネーム空間の例示



- X.500に準拠する
- DCEディレクトリ・サービスはグｒ一羽るネーム空間と，"..." に対し，同一のインタフェースを提供する。
- ... ローカルなネーム空間,mataha セルネーム空間
- /.: セルネーム空間のルート
- /: セル・ファイル・システムのルート

6.3 DCE セル・ディレクトリ・サービス機能

- クライアント側でキャッシングを行う特殊目的のデータベース
- スケーラビキティを実現するための分割管理
- キャッシュとデータベースのチェックポイントを使った回復機能
- RPC認証とアクセス制御リスト(ACL)を仕様したセキュリティ
- 分散かした管理機構

7. OSF/DFS - 分散ファイルシステム

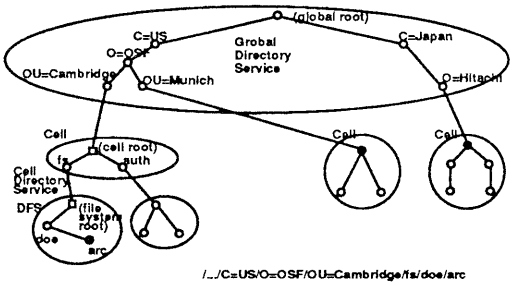
7.1 基本概念

- ファイルへのアクセスの共有を容易にする
- 物理的なロケーションに依存しない
- 統一的なファイル・パス名
- ディスクレス・ワークステーションをサポートする

7.2 DFSの目標

- データへの統一的なアクセスを実現するために，X.500と統合化されている。

図9 グローバルネーム空間



- 「トークン」による制御
- 使いやすさ
- 機能
  - -- 透過性
    - = ただし，管理者は問題・障害の特定のために透過性を「破る」事ができる
  - -- レプリケーションによる高い可用性
    - = レプリケーションDFSの統合化された構成要素としてサポートされる。
  - -- 容易な管理
  - -- ログ管理にもとづくローカル・ファイル

・システムによる高信頼性
-- ローカル・キャッシングによる高性能
-- スケーラビキティ
-- ユーザ定義によるアクセス制御，案ぜんなRPCの上にセキュリティを構築
-- 他のファイルシステム，特にNFSとの相互運用性をプロトコル・エクスポータで実現

- DFSクライアント
  -- 基本セットは全てのVFSで動作する（全てのUNIXファイル・システム）
  -- 拡張ファイル・サーバはDFSローカル・ファイル・システムを必要とする。。

7.3 トークン制御

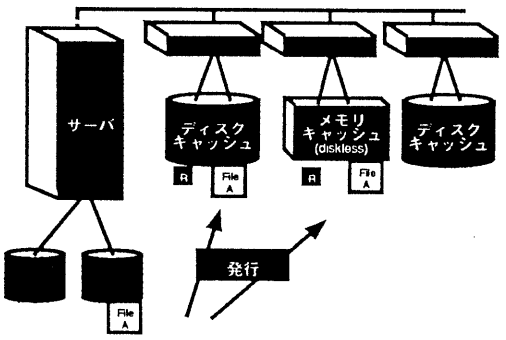
- 「トークン」はDSFサーバーが制御し，クライアントはトークンの受け取りを拒否できない。
- サーバからクライアントへの「約束」
- 単一サイト・セマンティックスの構築
- 最小限の通信
- 定常状態ではトラフィックが不要（ディレクトリ更新を除く）
- 他のユーザに許可を与える場合は取り消される
- バイト幅を指定する事が可能

7.4 トークン規則

トークンは次の規則に従う。

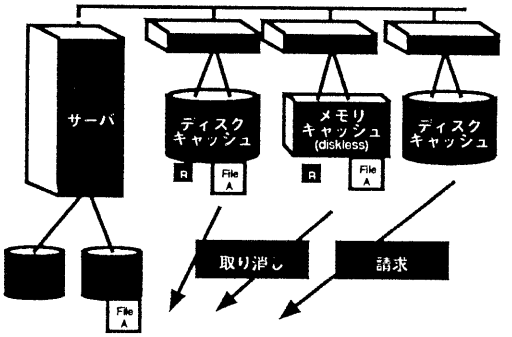
- 型は互いに独立している。
- １つの型について，書き込みを行う "writer" は一人。
- 読みとりを行う" Reader" と"Writer"は Non-Compatible である。
- 読みとりを行う複数の"Reader"は互いに Compatible である。

図 10 読みとリトークンの発行



- 図11はユーザ２人がファイルをオープンしている。読みとリトークンが発行される。

図 11 書き込みトークンの発行

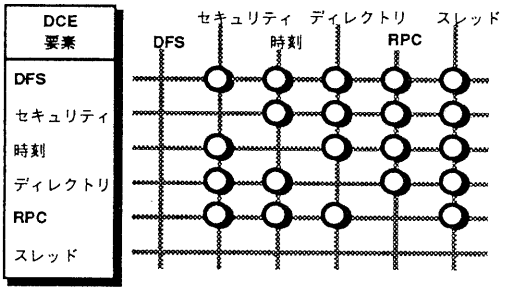


- 上図は一人のユーザがオープンされているファイルに書き込みを要求する時
- サーバハトークンを取り消し，書き込みトークンを発行する。

8. まとめ

OSF/DCEは相互に協調動作をする自律的な構成要素からなってるが，個々の物の並存ではないという当初の目標を実現したことを次の図によって示す。

図 12 OSF/DCE統合マトリックス



個別の機能の単なる集積を越えた，統合化分散環境として，今後のオブジェクト指向環境においてもインフラストラクチャとして，広く利用されて行くと思われる。